МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ Федеральное государственное бюджетное образовательное учреждение высшего образования **«Вятский государственный университет»**

**(ФГБОУ ВО «ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Лабораторный практикум

по дисциплине

«Технологии программирования»

Выполнила студентка группы ИВТ-31\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Седов М.Д./

Проверил преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л./

Киров 2019

Введение

Современный мир наполнен информацией вокруг нас. Анализы, статистики, сбор различных данных – все это представляет собой огромные ресурсы, сложные для восприятия и понимания.

В настоящее время большой популярностью пользуются компьютерные и мобильные приложения, позволяющие быстро получать текущую информацию по выполнению показателей своих договоров.

Одна из задач подобных сервисов – предоставить пользователю максимально полную и точную информацию о показателях, также изготовители добавляют к основным возможностям программных продуктов множество других разнообразных по своему содержанию функций. В настоящее время данные, отражающие состояние каждого показателя по трудовому договору, не работают в режиме реального времени. Так же необходимо обрабатывать информацию, которая в дальнейшем доносится до факультетов, кафедр и т.д.

Кроссплатформенная разработка, направленная на мобильную платформу и на персональные компьютеры под операционной системой Windows 10 будет иметь схожий интерфейс, и пользователем будет интуитивно понятно, как работать при переходе на другую ОС. Таким образом, приложение, отображающее статусы показателей каждого трудового договора, заинтересует сотрудников университета.

1. **Разработка технического задания на создание программного обеспечения.**

В данном разделе рассматриваются основные требования к разрабатываемой структуре.

* 1. Обоснование темы разработки. Основанием для разработки является учебный план кафедры Электронных вычислительных машин.
  2. Краткая характеристика области применения.На основе разрабатываемой структуры в дальнейшем будет реализована программа, планируемая к использованию на мобильных устройствах под управлением ОС Android/IOS.
  3. Назначение разработки включает в себя два направления:
* Функциональное назначение – структура должна с требуемой степенью детализации характеризовать выполнение эффективного контракта у сотрудника университета.

1.4 Требования к структуре по следующим критериям:

* + 1. Требования к составу выполняемых функций:

– Проверка логина и пароля пользователя при входе на наличие в базе.

– Возможность просмотра трудовых договоров по сотруднику университета ВятГУ.

– Возможность просмотра всех показателей по выбранному договору (так же по показателю видно плановое значение и фактическое значение в режиме реального времени).

– При нажатии на некоторый показатель открывается новая страница, в которой доступен детальный анализ показателя.

– Вывод всех договоров текущего пользователя в табличной форме с возможностью перехода по нему.

– Вывод всех показателей по выбранному договору в табличной форме с детальной информацией по каждому из них.

* + 1. Входными данными для ПО являются события нажатий пользователем в разных частях экрана телефона.
    2. Требования к организации выходных данных – на выходе приложение предполагает визуализацию конкретной страницы с текстовой составляющей.
    3. Требования к составу и параметрам технических средств – размер оперативной памяти: не менее 256 Мбайт, требуемый объем памяти для хранения данных: не менее 256 Мбайт.
  1. Требования к проектной документации. Состав проектной документации должен включать в себя:
* Техническое задание;
* Технический проект.
  1. Структурирование точек зрения. Данный этап предполагает создание иерархии сгруппированных точек зрения. При формировании требований их необходимо идентифицировать. На рисунке 1 приведена диаграмма идентификации точек зрения. На рисунке 2 показана часть иерархии точек зрения для разрабатываемой системы.
  2. **Анализ опорных точек зрения**

Диаграмма идентификации точек зрения представлена на рисунке 1, диаграмма иерархии точек зрения представлена на рисунке 2.

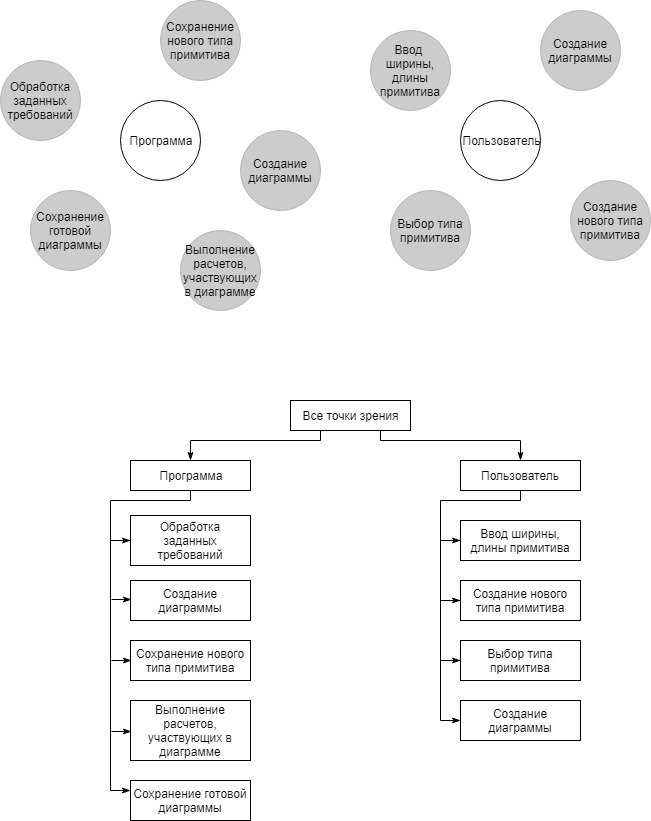


Рисунок 1 - Диаграмма идентификации точек зрения

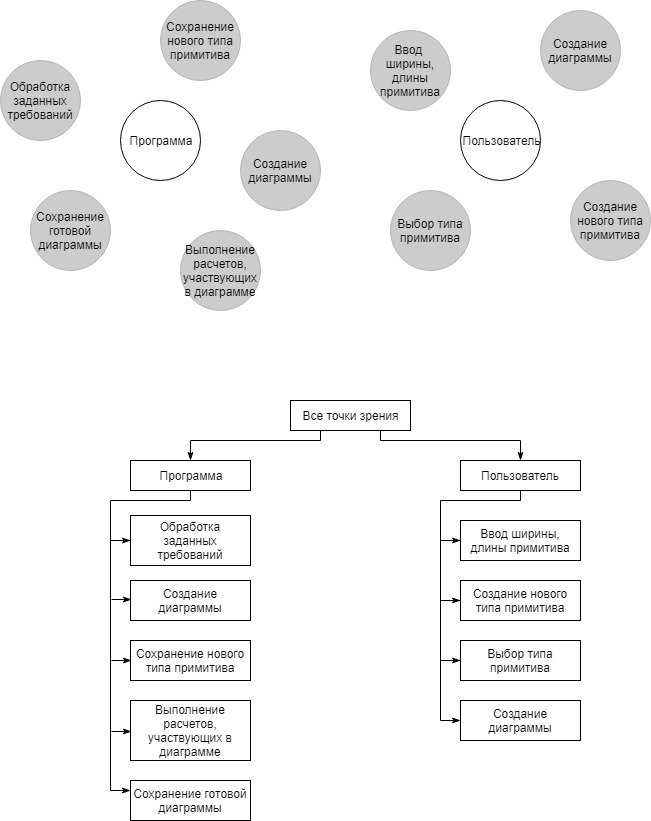


Рисунок 2 – Диаграмма иерархии точек зрения

1. **Структурный подход к проектированию программного обеспечения**

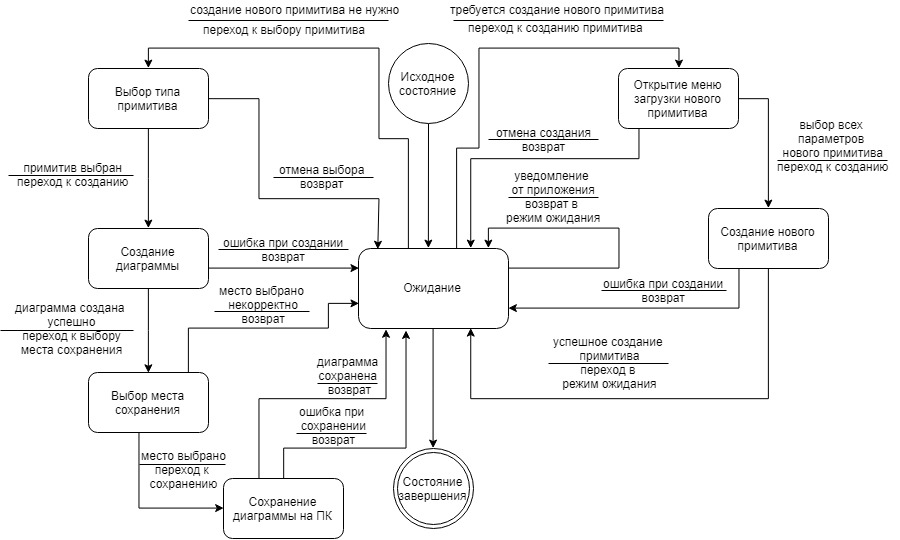


Рисунок 3 - Диаграмма переходов состояний

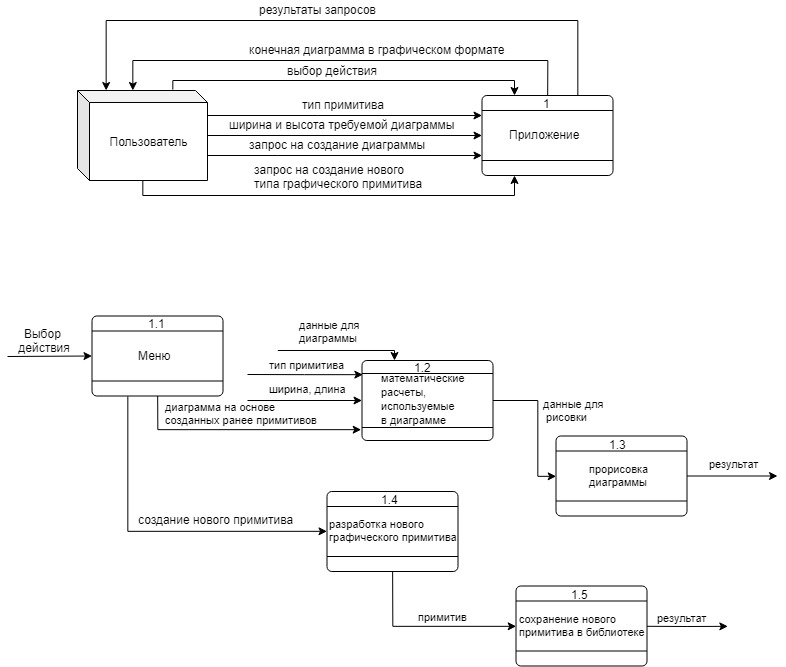


Рисунок 4 – Контекстная диаграмма потоков данных

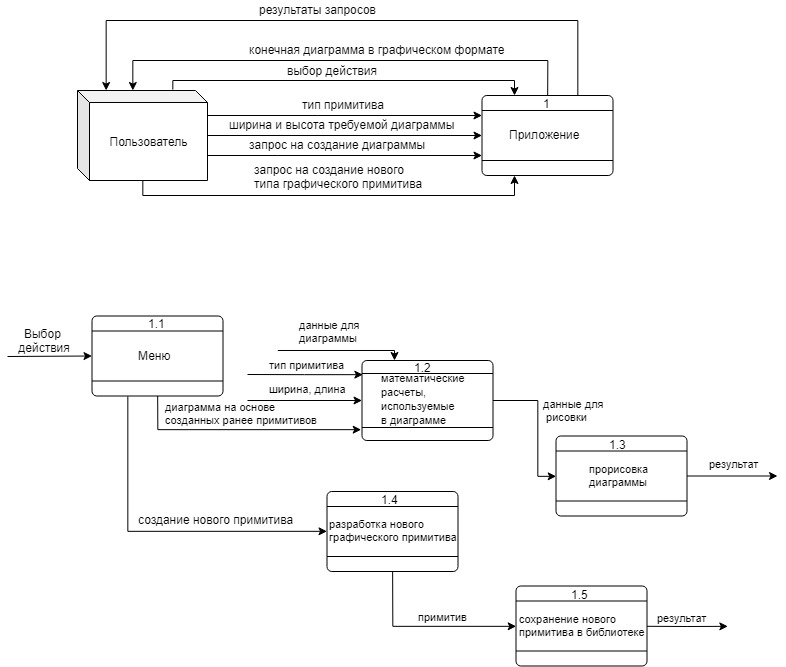


Рисунок 5 – Детализирующая диаграмма потоков данных

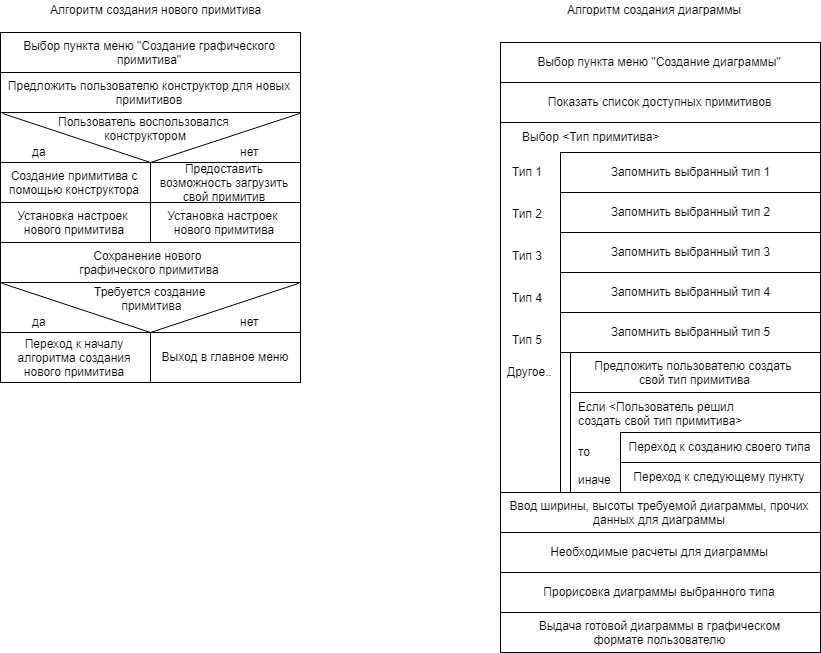


Рисунок 6 – Диаграмма Несси-Шнейдермана

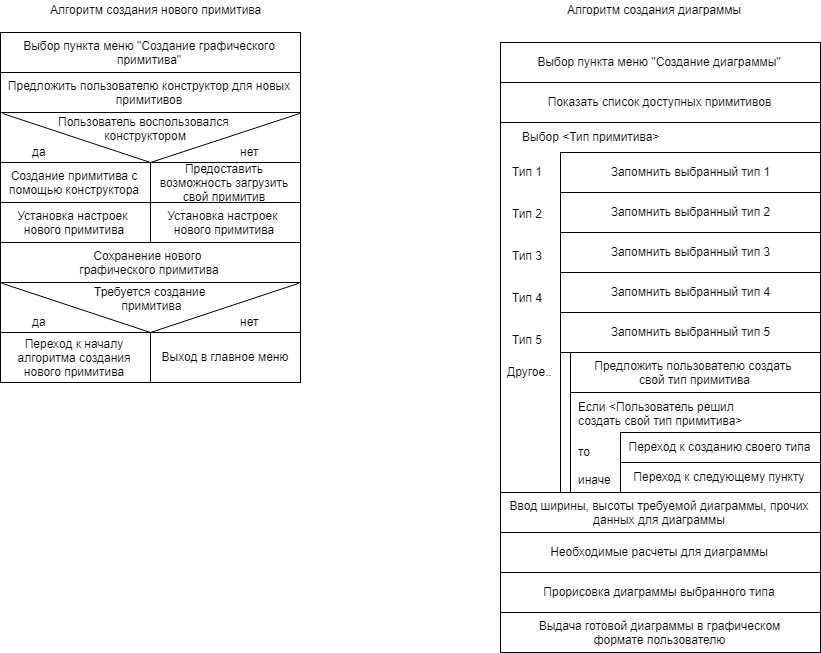


Рисунок 7 – Flow-форма

1. **Реализация программного обеспечения**

В данном лабораторном практикуме был написан код реализации фабричного метода разрабатываемой структуры, так как этот метод является наиболее сложным и интересным для изучения.

Фабричный метод (англ. *Factory Method)* – порождающий шаблон проектирования, предоставляющий подклассам (дочерним классам) интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Используется, когда:

* классу заранее неизвестно, объекты каких подклассов ему нужно создавать.
* класс спроектирован так, чтобы объекты, которые он создаёт, специфицировались подклассами.
* класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и планируется локализовать знание о том, какой класс принимает эти обязанности на себя

В разрабатываемой структуре данный шаблон используется потому, что для каждого типа примитива требуются определенные входные данные и вычисления. Интерфейсный класс «Keeper» хранит в себе данные для построения диаграмм из всех графических примитивов. Класс «Data», обращаясь к нему, получает на вход данные, требуемые определенному типу примитива. Таким же образом из класса «Primitive» вызывается метод вычислений, необходимых для построения диаграммы, интерфейсного класса «Drawer».

Листинг программы:

**Класс main**

package factory;

import java.util.Scanner;

public class main {

public static void main(String[] args) {

System.***out***.printf("1. Круговая диаграмма \n2. Столбчатая диаграмма\n3. Горизонтальная диаграмма\n4. Количественная диаграмма\n");

Scanner in = new Scanner(System.***in***);

System.***out***.print("Input a number: ");

int num = in.nextInt();

switch(num)

{

case 1 :

{ Scanner in1 = new Scanner(System.***in***);

System.***out***.printf("Введите число компонентов \n");

int numCircle = in1.nextInt();

if(in1.hasNextInt()) {

numCircle = in1.nextInt();

} else {

System.***out***.print("Вы ввели не целое число");

break;

}

Scanner in2 = new Scanner(System.***in***);

int array1[] = new int[numCircle]; //

System.***out***.print("Введите размерность компонентов:\n");

for (int i = 0; i < numCircle; i++) {

if(in2.hasNextInt()) {

array1[i] = in2.nextInt();

} else {

System.***out***.print("Вы ввели не целое число\n");

break;

}

}

System.***out***.printf("Введите подписи диаграммы:\n");

String[] array2 = new String[numCircle];

Scanner in3 = new Scanner(System.***in***);

String input;

for (int i = 0; i < numCircle; i++) {

input = in3.nextLine();

array2[i] = input;}

in3.close();

in1.close();

in2.close();

Drawer myCircleFactory = new CircleCompute();

keeper myCircleDiagram = myCircleFactory.Computation();

System.***out***.println(myCircleDiagram);

break;

}

case 2 :{

Scanner in4 = new Scanner(System.***in***);

System.***out***.printf("Введите число компонентов \n");

int numScale1 = in4.nextInt();

if(in4.hasNextInt()) {

numScale1 = in4.nextInt();

} else {

System.***out***.print("Вы ввели не целое число");

break;

}

Scanner in5 = new Scanner(System.***in***);

int array3[] = new int[numScale1];

System.***out***.print("Введите размерность компонентов:\n");

for (int i = 0; i < numScale1; i++) {

if(in4.hasNextInt()) {

array3[i] = in4.nextInt();

} else {

System.***out***.print("Вы ввели не целое число\n");

break;

}

}

System.***out***.printf("Введите подписи диаграммы:\n");

String[] array4 = new String[numScale1];

Scanner in6 = new Scanner(System.***in***);

String input;

for (int i = 0; i < numScale1; i++) {

input = in6.nextLine();

array4[i] = input;}

in6.close();

in4.close();

in5.close();

Drawer myColumnarFactory = new ColumnarCompute();

keeper myColumnarDiagram = myColumnarFactory.Computation();

System.***out***.println(myColumnarDiagram);

break;

}

case 3 :

{

Scanner in7 = new Scanner(System.***in***);

System.***out***.printf("Введите число компонентов \n");

int numScale2 = in7.nextInt();

if(in7.hasNextInt()) {

numScale2 = in7.nextInt();

} else {

System.***out***.print("Вы ввели не целое число");

break;

}

Scanner in8 = new Scanner(System.***in***);

int array5[] = new int[numScale2]; //

System.***out***.print("Введите размерность компонентов:\n");

for (int i = 0; i < numScale2; i++) {

if(in8.hasNextInt()) {

array5[i] = in8.nextInt();

} else {

System.***out***.print("Вы ввели не целое число\n");

break;

}

}

System.***out***.printf("Введите подписи диаграммы:\n");

String[] array6 = new String[numScale2];

Scanner in9 = new Scanner(System.***in***);

String input;

for (int i = 0; i < numScale2; i++) {

input = in9.nextLine();

array6[i] = input;}

in9.close();

in8.close();

in7.close();

Drawer myHorizontalFactory = new HorizontalCompute();

keeper myHorizontalDiagram = myHorizontalFactory.Computation();

System.***out***.println(myHorizontalDiagram);

break;

}

case 4 :

{

Scanner in10 = new Scanner(System.***in***);

System.***out***.printf("Введите начальное число \n");

int baseComp = in10.nextInt();

if(in10.hasNextInt()) {

baseComp = in10.nextInt();

} else {

System.***out***.print("Вы ввели не целое число");

break;

}

Scanner in11 = new Scanner(System.***in***);

System.***out***.printf("Введите число врменных промежутков \n");

int numTime = in11.nextInt();

if(in11.hasNextInt()) {

numTime = in11.nextInt();

} else {

System.***out***.print("Вы ввели не целое число");

break;

}

Scanner in12 = new Scanner(System.***in***);

int array7[] = new int[numTime];

System.***out***.print("Заполните массив прироста:\n");

for (int i = 0; i < numTime; i++) {

if(in12.hasNextInt()) {

array7[i] = in12.nextInt();

} else {

System.***out***.print("Вы ввели не целое число\n");

break;

}

}

System.***out***.printf("Введите подписи временных промежутков:\n");

String[] array8 = new String[numTime];

Scanner in13 = new Scanner(System.***in***);

String input;

for (int i = 0; i < numTime; i++) {

input = in13.nextLine();

array8[i] = input;}

in10.close();

in11.close();

in12.close();

in13.close();

Drawer myQuantiFactory = new QuantiCompute();

keeper myQuantiDiagram = myQuantiFactory.Computation();

System.***out***.println(myQuantiDiagram);

break;

}

default: break;

}

in.close();

}

}

**Описание классов**

package factory;

import java.util.ArrayList;

import java.util.List;

public abstract class keeper {

String type;

List <String> dataType = new ArrayList();

public String toString(){

return "Type of primitive: " + type + "\n" + dataType;

}

}

abstract class Drawer{

public abstract keeper Computation();

}

class ColumnarCompute extends Drawer{

public keeper Computation(){

return new ColumnarDiagram();

}

}

class CircleCompute extends Drawer{

public keeper Computation(){

return new CircleDiagram();

}

}

class HorizontalCompute extends Drawer{

public keeper Computation(){

return new HorizontalDiagram();

}

}

class QuantiCompute extends Drawer{

public keeper Computation(){

return new QuantiDiagram();

}

}

class ColumnarDiagram extends keeper{

public ColumnarDiagram(){

type = "Columnar Diagram";

dataType.add("numScale");

dataType.add("array numScaleComp");

dataType.add("array nameScale");

}

}

class CircleDiagram extends keeper{

public CircleDiagram(){

type = "Circle Diagram";

dataType.add("numCircle");

dataType.add("array numCircleComp");

dataType.add("array nameCircle");

}

}

class HorizontalDiagram extends keeper{

public HorizontalDiagram(){

type = "Horizontal Diagram";

dataType.add("numScale");

dataType.add("array numScaleComp");

dataType.add("array nameScale");

}

}

class QuantiDiagram extends keeper{

public QuantiDiagram(){

type = "Quanti Diagram";

dataType.add("baseComp");

dataType.add("numTime");

dataType.add("array nameGrowth");

dataType.add("array nameTime");

}

}

1. **Тестирование программ**

4.1 Тестирование по принципу «белого ящика»

* Метод покрытия операторов

Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз. Результаты тестирования данным методом в таблице 4.1.1

Таблица 4.1.1 – Тестирование методом покрытия операторов

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Ожидаемый результат | Фактический результат | Результат тестирования |
| Выбор пункта меню 1  Число компонентов 2  Размерность компонентов 23, 34  Подписи диаграммы first, second | Выдача типа круговой диаграммы и ее аргументов | Type of primitive: Circle Diagram  [numCircle, array numCircleComp, array nameCircle] | Успешно |
| Выбор пункта меню 2  Число компонентов 1  Размерность компонентов семь  Подписи диаграммы first | Выдача ошибки | Выдача ошибки | Успешно |
| Выбор пункта меню 3  Число компонентов два  Размерность компонентов 2, 7  Подписи диаграммы first, second | Выдача ошибки | Выдача ошибки | Успешно |
| Выбор пункта меню 4  Начальное число 34  Число временных промежутков 2  Массив прироста 2, 7  Подписи временных промежутков first, second | Выдача типа количественной диаграммы и ее аргументов | Type of primitive: Quanti Diagram  [baseComp, numTime, array nameGrowth, array nameTime] | Успешно |
| Выбор пункта меню первый  Число компонентов 2  Размерность компонентов 2, 7  Подписи диаграммы first, second | Выдача ошибки | Выдача ошибки | Успешно |

* Метод покрытия решений

Согласно методу покрытия решений каждое направление пе­рехода должно быть реализовано, по крайней мере, один раз. Так как этот метод включает в себя критерий покрытия операторов, так как при выполнении всех направлений переходов выполнятся все операторы, проверим этим методом только те пути, которые не проверены методом покрытия операторов.

Все ветки программы помечены буквами для удобства обозначения путей (Рисунок 4.1). Результат тестирования приведен в таблице 4.1.2

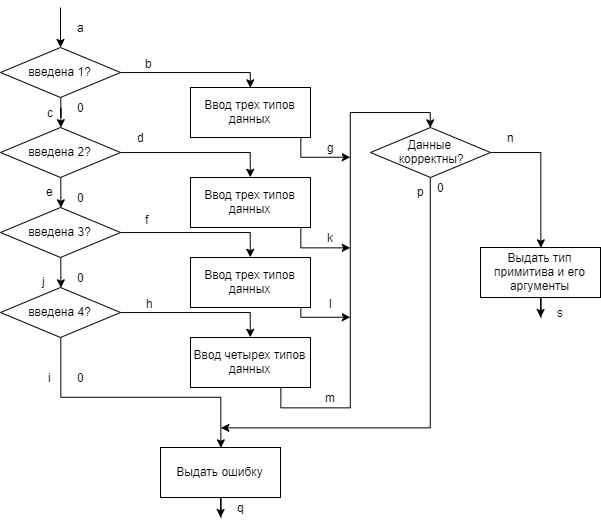


Рисунок 4.1 – Обозначение веток программы буквами

Таблица 4.1.2 – Метод покрытия решений

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Ожидаемый результат | Фактический результат | Результат тестирования |
| abgpq | Выдача ошибки из-за некорректного ввода данных | Выдача ошибки из-за некорректного ввода данных | Успешно |
| acdkns | Выдача столбчатого типа диаграммы с его аргументами | Type of primitive: Columnar Diagram  [numScale, array numScaleComp, array nameScale] | Успешно |
| aceflns | Выдача горизонтального типа диаграммы с его аргументами | Type of primitive: Horizontal Diagram  [numScale, array numScaleComp, array nameScale] | Успешно |

**Вывод:**

В ходе тестирования по принципу «белого ящика» все тесты были пройдены успешно.

4.2 Тестирование по принципу «Черного ящика»

Для тестирования по данному принципу воспользуемся методом эквивалентного разбиения.

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа:

1. выделение классов эквивалентности
2. построение тестов.

4.2.1 Выделение классов эквивалентности. Разбиение на классы представлено в таблице 4.2.1.1

Таблица 4.2.1.1 – Выделение классов эквивалентности

|  |  |  |  |
| --- | --- | --- | --- |
| Входные условия | | Правильные классы эквивалентности | Неправильные классы эквивалентности |
| Ввод пункта меню | | 1-4(1) | <1,(25) >4(2) |
| Условия для столбчатой диаграммы | Число столбцов | Целое число(3) | Не целое число(4) |
| Массив высот столбцов | Массив целых чисел(5) | Массив имеет что-то иное(6) |
| Массив названий столбцов | Массив строк(7) | - |
| Условия для круговой диаграммы | Число секторов | Целое число(8) | Не целое число(9) |
| Массив толщины секторов | Массив целых чисел(10) | Массив имеет что-то иное(11) |
| Массив названий секторов | Массив строк(12) | - |
| Условия для горизонтальной диаграммы | Число шкал | Целое число(13) | Не целое число(14) |
| Массив длины шкал | Массив целых чисел(15) | Массив имеет что-то иное(16) |
| Массив названий шкал | Массив строк(17) | - |
| Условия для количественной диаграммы | Начальное число компонентов | Целое число(18) | Не целое число(19) |
| Число временных промежутков | Целое число(20) | Не целое число(21) |
| Массив прироста | Массив целых чисел(22) | Массив имеет что-то иное(23) |
| Массив названий временных промежутков | Массив строк(24) | - |

4.2.2 Построение тестов и непосредственное тестирование. Тесты для правильных классов эквивалентности приведены в следующей таблице.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название теста | Покрываемые классы | Способ проверки | Комментарий | Результат |
| Проверка ввода целых чисел | (3), (8), (13), (18), (20) | Ввод целых чисел | Программа принимала ввод | Успешно |
| Проверка ввода массивов целых чисел | (5),(10),(15),(22) | Ввод целых чисел в качестве элементов массива | Программа принимала ввод | Успешно |
| Проверка ввода массива строк | (7),(12),(17),(24) | Ввод целых, дробных чисел, знаков, символов на место элементов массива | Программа принимала ввод | Успешно |
| Проверка выбора пункта меню | (1) | Выбор пункта меню с 1 по 4 | Программа принимала ввод и открывала необходимый пункт | Успешно |

Неправильные классы эквивалентности и соответствующие им тесты приведены в следующей таблице.

|  |  |  |
| --- | --- | --- |
| Номер класса | Неправильный класс эквивалентности | Тест |
| (2) | >4 | Input (24) |
| (4) | Не целое число | Input (3,5) |
| (6) | Массив имеет что-то кроме целого числа | Input (lmnb) |
| (9) | Не целое число | Input (6,78) |
| (11) | Массив имеет что-то кроме целого числа | Input (dfg) |
| (14) | Не целое число | Input (3,45) |
| (16) | Массив имеет что-то кроме целого числа | Input (ва4) |
| (19) | Не целое число | Input (Ляля) |
| (21) | Не целое число | Input (Не целое число) |
| (23) | Массив имеет что-то кроме целого числа | Input (н7) |
| (25) | <1 | Input (0) |

**Вывод:**

В ходе тестирования по принципу «черного ящика» методом эквивалентного разбиения все тесты были пройдены успешно.

1. **Методология объектно-ориентированного моделирования**

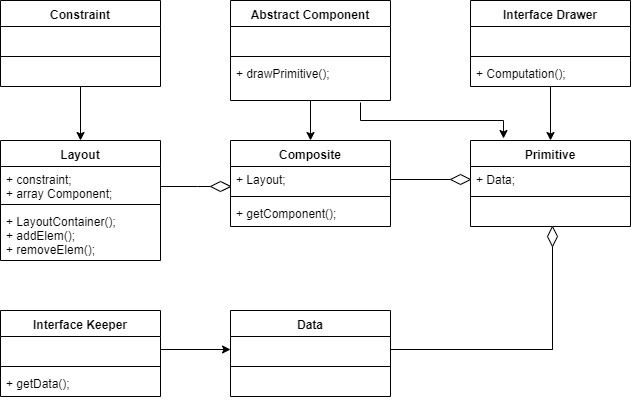
****

Рисунок 8 – Диаграмма классов UML

Таблица 5.1 – Описание классов диаграммы, их атрибутов и методов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя класса | Тип класса | Атрибуты | Методы | Комментарий |
| Component | Abstract |  | drawPrimitive() | Определяет интерфейс для простых компонентов (класс Primitive) и составных (класс Composite). |
| Composite |  | layout | getComponent() | Класс-контейнер, содержащий дочерние элементы, но ничего не знающий об их типах (простые они или составные). |
| Primitive |  | data |  | Простой компонент структуры, не имеет ответвлений и не передает никому управление. |
| Drawer | Interface |  | computation() | Интерфейсный класс, который содержит в себе методы вычислений, необходимых для любого вида диаграмм. |
| Keeper | Interface |  | getData() | Интерфейсный класс, который хранит в себе данные всех типов примитивов. |
| Data |  |  |  | Класс данных, который получает на вход данные класса Keeper для определенного типа примитива. Экземпляр данного класса используется в классе Primitive в качестве аргумента. |
| Constraint |  |  |  | Класс, который хранит в себе ограничения данных для всех типов примитивов. |
| Layout |  | constraint  array Component | layoutContainer()  addElem()  removeElem() | Менеджер расположения, который определяет в каком порядке примитивы располагаются на диаграмме. |

1. **Методология управления проектами**
2. Построение модели управления проектом

T1 – Обсуждение проекта с заказчиком

Т2 – Подготовка плана проекта

Т3 – Разработка технического задания

Т4 – Обзор аналогов

Т5 – Структурное, объектно-ориентированное проектирование

Т6 – Реализация метода программы

Т7 – Проверка структуры на конкретных типах примитивов

Т8 – Подготовка документации

Длительность и зависимость всех этапов указана в таблице 6.1

Таблица 6.1 – Этапы проекта

|  |  |  |
| --- | --- | --- |
| Этап | Длительность (дни) | Зависимость |
| Т1 | 1.09-15.09 (14) |  |
| Т2 | 16.09-30.09 (14) | Т1 |
| Т3 | 1.10-21.10(20) | Т1(М1) |
| Т4 | 22.10-06.11 (14) | Т3 |
| Т5 | 7.11-28.11 (21) | Т2,Т4 |
| Т6 | 29.11-14.12 (15) | Т5 |
| Т7 | 15.12-20.12 (6) | Т6, Т5(М2) |
| Т8 | 21.12-31.12 (10) | Т7 |

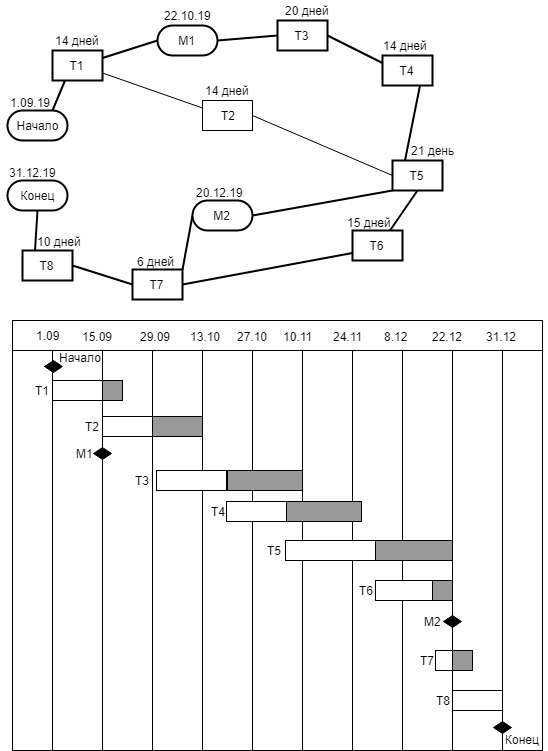


Рисунок 9 – Сетевая диаграмма этапов

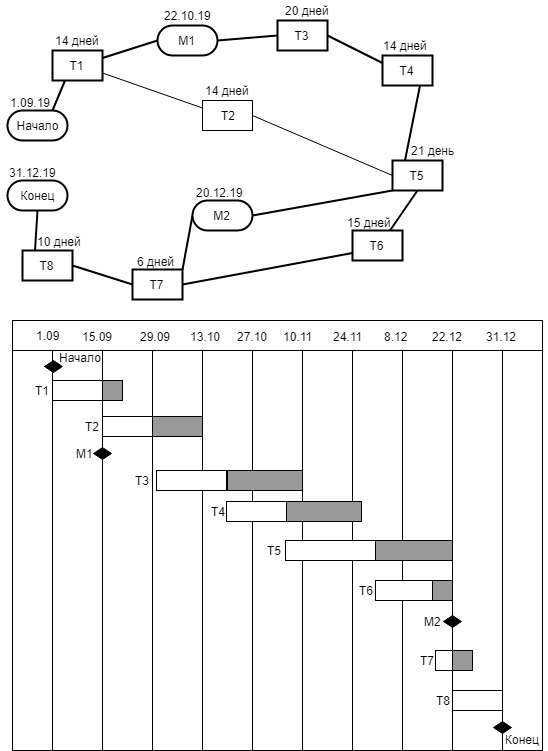


Рисунок 10 – Временная диаграмма длительности этапов

1. Возможные риски
   1. Описание возможных рисков

|  |  |  |
| --- | --- | --- |
| Риск | Тип риска | Описание риска |
| Изменение требований | Риск для проекта и разрабатываемого продукта | Заказчик поменяет свои требования |
| Задержка в разработке | Риск для проекта и разрабатываемого продукта | Разработчик не укладывается во временные рамки |
| Появление конкурирующего продукта | Бизнес-риск | Снижение потребности в разрабатываемом продукте |

* 1. Анализ возможных рисков

|  |  |  |
| --- | --- | --- |
| Риск | Вероятность | Степень ущерба |
| Изменение требований | Средняя | Серьезная |
| Задержка в разработке | Высокая | Серьезная |
| Появление конкурирующего продукта | Средняя | Серьезная |

* 1. Стратегия планирования рисков

|  |  |
| --- | --- |
| Риск | Стратегия |
| Изменение требований | Попытаться определить требования, наиболее вероятно подверженные изменениям; в структуре системы не отображать детальную информацию |
| Задержка в разработке | Рассмотреть вопрос о покупке системных компонентов, исследовать возможность использования генератора программного кода |
| Появление конкурирующего продукта | Оценить актуальность проблемы, произвести обзор аналогов, повысить уникальность и удобство использования продукта |